

Fatui Organizational System

1. User requirements

Fatui needs a system to manage employees, missions, and store important information about regions, their leaders, and gods. This will help in organizing operations and accessing critical data easily.

System features:

1. Employee management

- Agents: Store details including name, role, and date of joining.
- Harbingers: Store details including name, title, number (1 to 11), region of origin, and date of joining.
- Each Harbinger has a group of agents serving them, with some agents supervising others who joined recently.

2. Mission management

- Create missions with details such as start date, deadline, description, and fund.
- Fund calculation is based on the rank of assigned Harbingers.
- Each mission must have at least one Harbinger assigned.
- Assign employees to missions.

3. Region information

- Store details of regions including name, element (e.g., wind, water), and a short description.
- Regions are governed by one or more leaders, with details including name and title.
- Each region worships a god, with details including name, element, list of titles, and age.

User interface requirements:

1. Mission overview and details

- View a list of all missions.
- Click on a mission to see details and assigned employees.
- Create new missions with relevant details.
- Assign employees to missions.

2. Employee assignment

- Assign and reassign agents to Harbingers and missions.
- Manage the supervision structure within Harbinger groups.

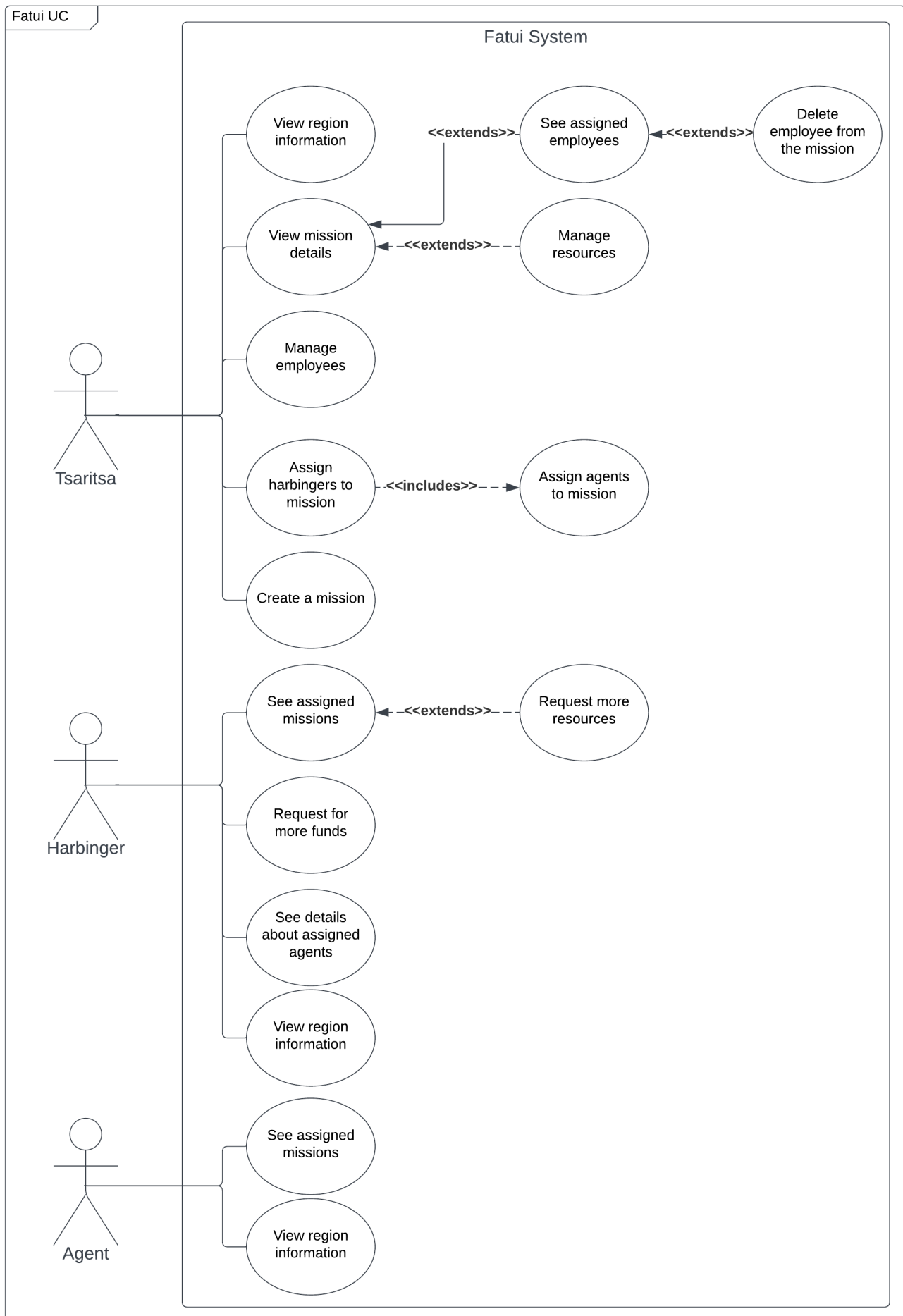
3. Region and leadership information

- View details about regions, their leaders, and gods.

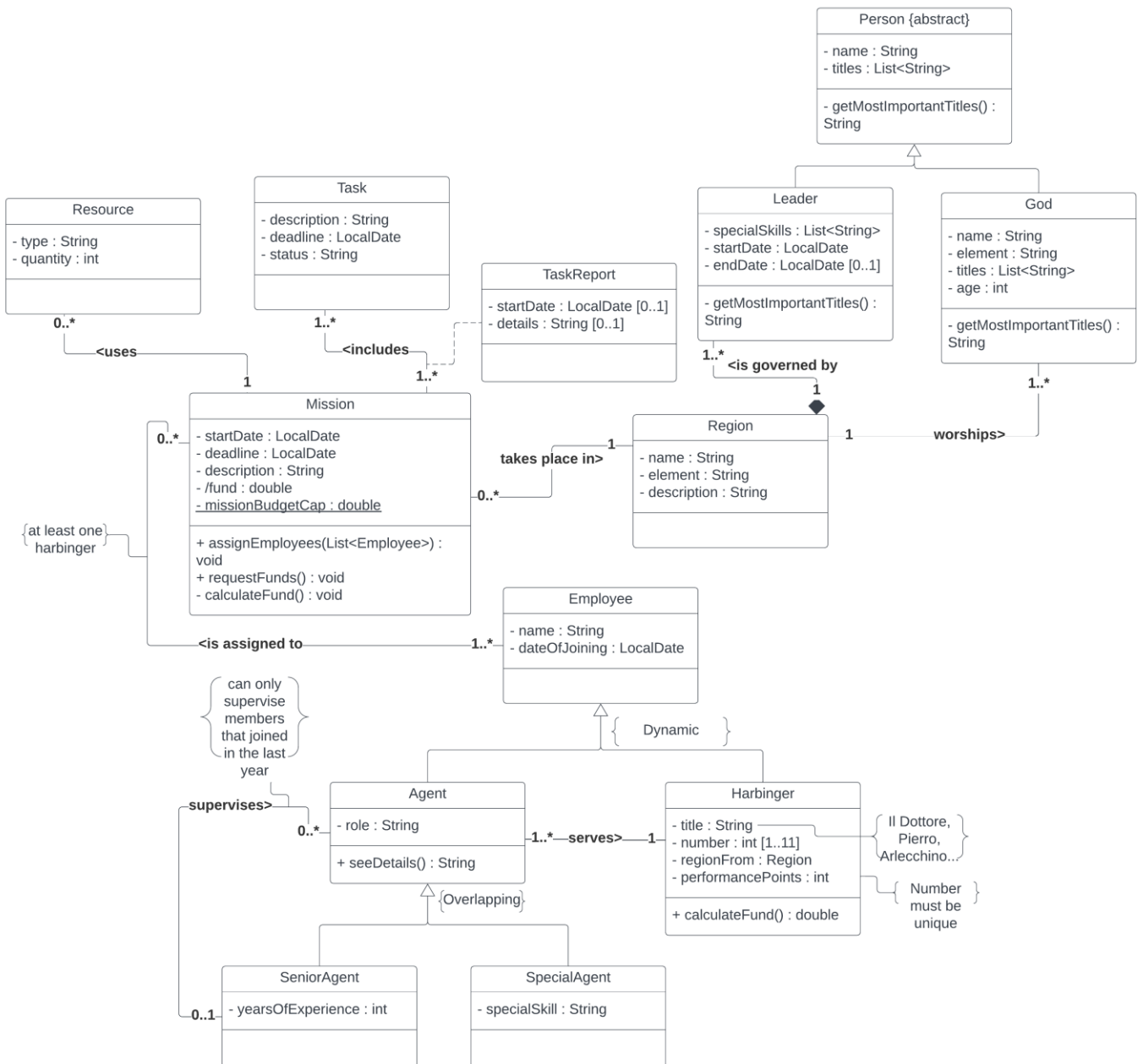
4. Resource management

- Track and manage resources needed for missions.

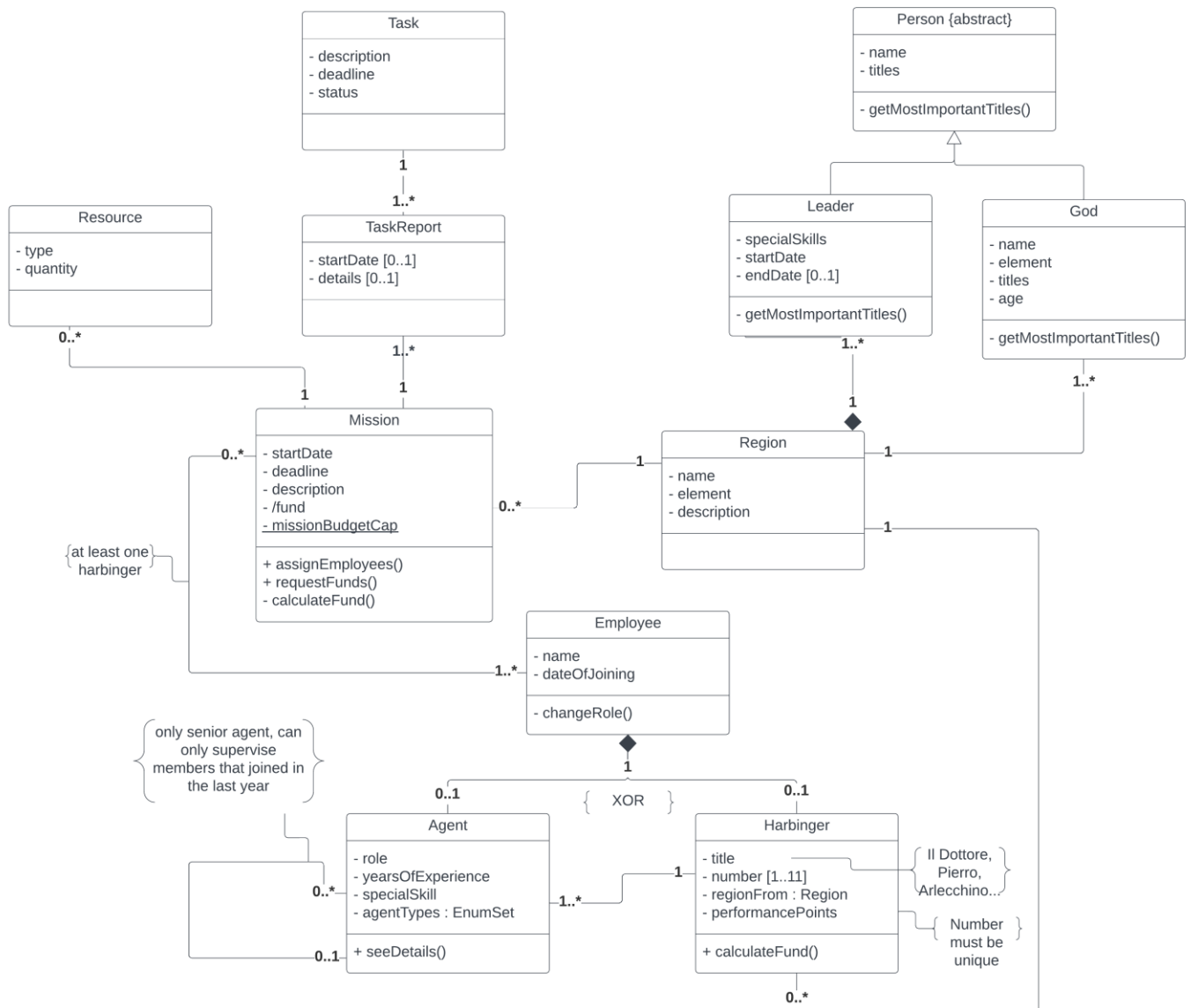
2. Use case diagram



3. Analytical class diagram



4. Design class diagram



5. Scenario of “See assigned employees” use case

See assigned employees - Use case scenario

Actors:

Tsaritsa

Dependencies:

See mission details - extends

Delete employee from the mission - extends

Triggers:

Tsaritsa enters the mission details.

Pre-conditions:

1. Tsaritsa has logged into the system.
2. There is at least one mission.
3. Tsaritsa has access to the mission details.

Basic flow of events:

1. Tsaritsa opens mission management page in the system.
2. System shows a list of available missions.
3. Tsaritsa chooses a mission to view.
4. System displays detailed information about this specific mission.
5. Tsaritsa chooses a button to display employees assigned to this mission.
6. System shows a list of employees assigned to this mission.
7. Tsaritsa reviews the employee details.
8. If needed, Tsaritsa can delete an employee from the mission.
9. System asks for confirmation.
10. Tsaritsa confirms her choice.

Alternative flow of events:

There is no available mission:

- 2a1. System shows information about no available missions.

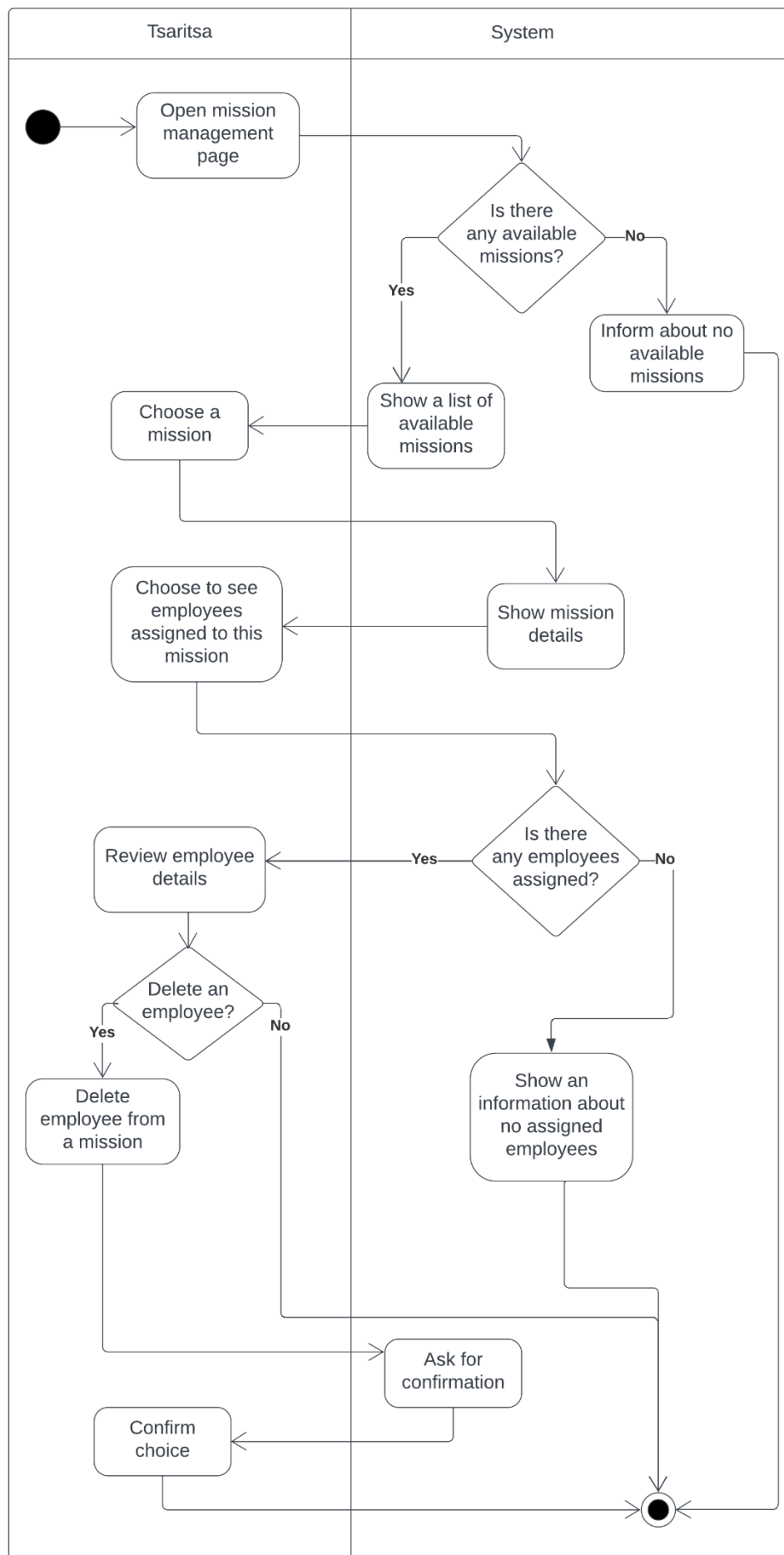
There is no employees assigned:

- 6a1. System shows information about no employees assigned.

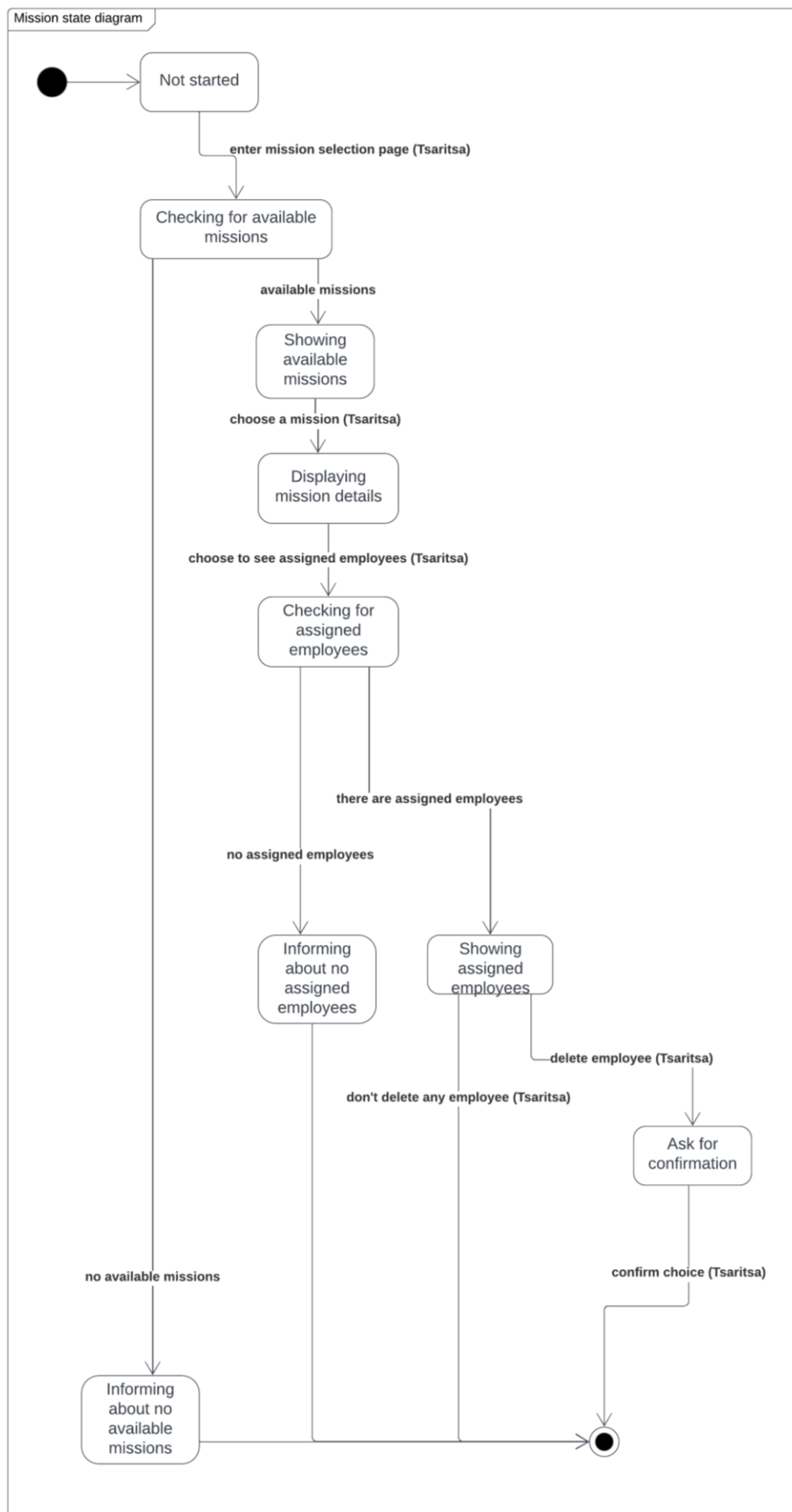
Post-conditions:

1. Selected employees are deleted from the mission.

6. Activity diagram of “See assigned employees” use case

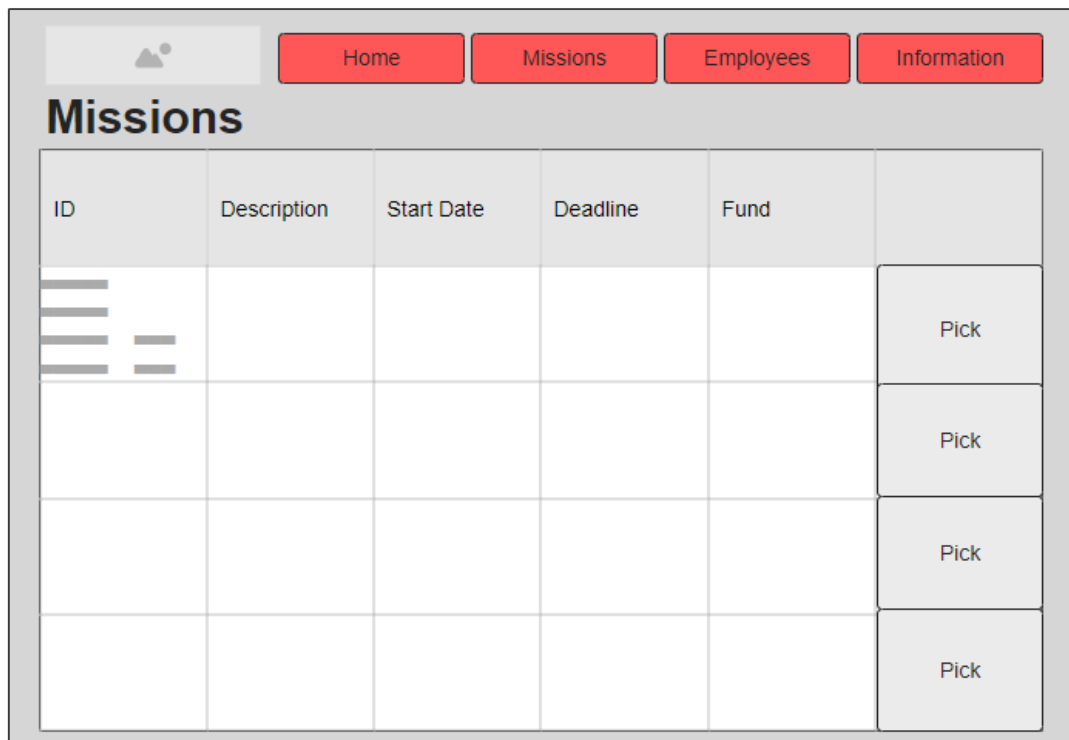


7. State diagram of “See assigned employees” use case



8. GUI design

Figure I. Mission list



The screenshot shows a web application interface for managing missions. At the top, there is a navigation bar with four red buttons: 'Home', 'Missions', 'Employees', and 'Information'. Below the navigation bar, the title 'Missions' is displayed in a large, bold font. The main content area contains a table with five columns: 'ID', 'Description', 'Start Date', 'Deadline', and 'Fund'. The table has four rows of data. Each row has a 'Pick' button in the rightmost column. The first row has a 'Pick' button, and the other three rows have 'Pick' buttons. The table is styled with a light gray background and white text.

ID	Description	Start Date	Deadline	Fund	
					Pick
					Pick
					Pick
					Pick

Figure II. No Available Missions Information

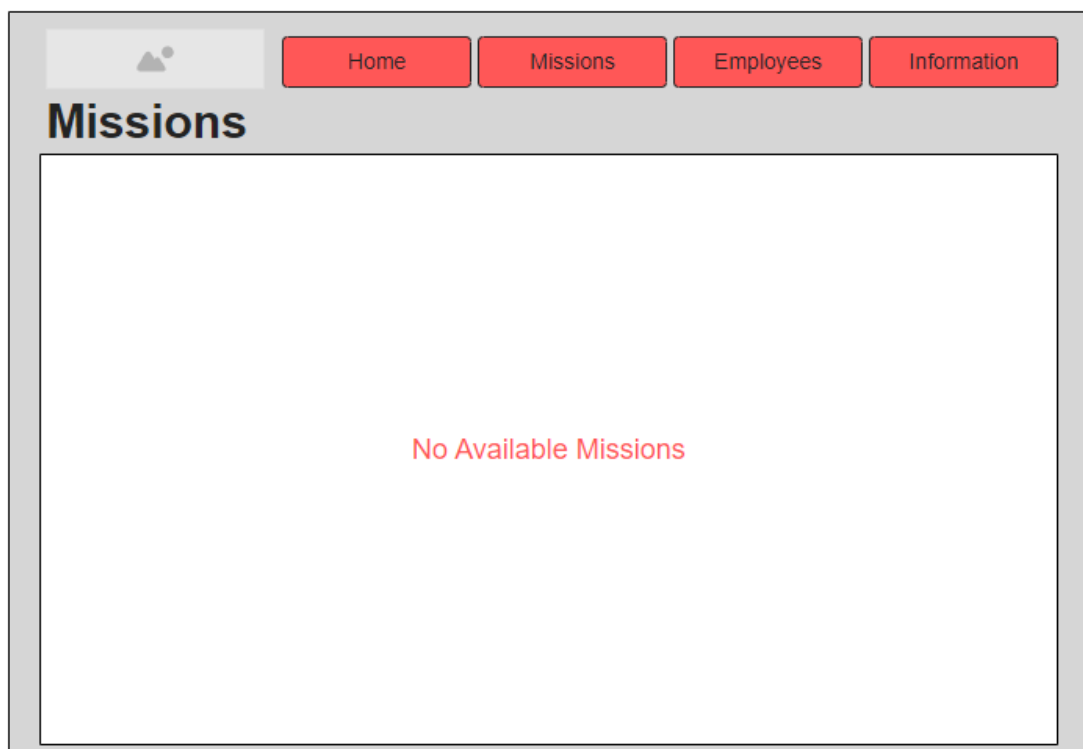


Figure III. Mission details page

Home

Missions

Employees

Information

Details

ID	Description	Start Date	Deadline	Fund
██████████				
██████████				
██████████				
██████████				
██████████				
██████████				
██████████				
██████████				
██████████				
██████████				

Resources

██████████

██████████

██████████

██████████

██████████

██████████

██████████

██████████

1

▲▼

1

▲▼

1

▲▼

1

▲▼

Employees

Show assigned employees

Figure IV. Showing assigned employees

Home

Missions

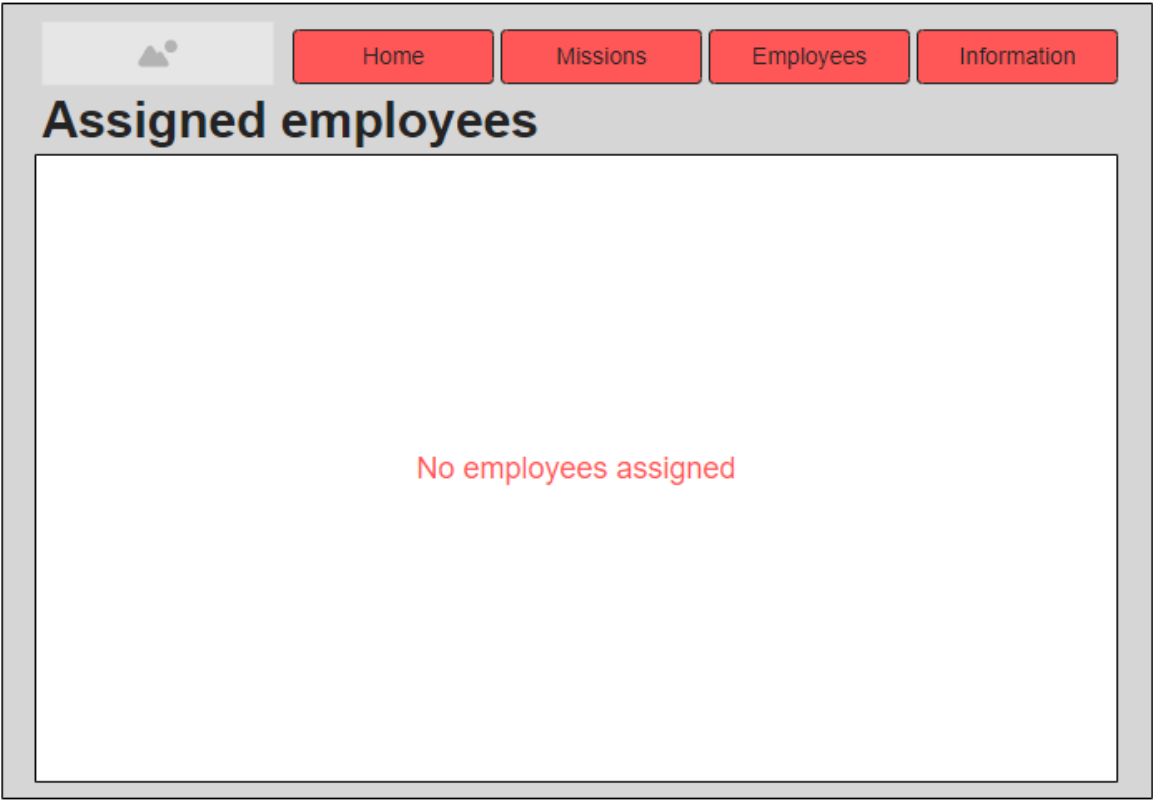
Employees

Information

Assigned employees

ID	Role	Other details ...	
██████████	Agent		Delete
	Harbinger		Delete
			Delete
			Delete

Figure V. No employees assigned information



9. Discussion

This project involves developing a system for managing employees, missions, and regions for the Fatui organization. The system includes various actors such as Tsaritsa, Harbingers, and Agents, each with specific roles and capabilities.

Dynamic analysis and design decisions:

1. Association with an Attribute:

To correctly implement an association with an attribute in a programming language, I created two associations between the classes and the attribute class TaskReport.

2. Derived attribute:

I added a function, calculateFund, to dynamically compute and manage the funds related to missions and resources.

3. Dynamic Inheritance to XOR Composition:

I changed the dynamic inheritance structure into an XOR composition. This decision simplifies the inheritance hierarchy, making it possible to implement the inheritance and ensuring the system is more maintainable by preventing entities from belonging to multiple conflicting types.

4. Role Change Function:

I implemented a changeRole() function to allow dynamic changes in the roles of employees. This function enables the system to change the specific employee's role over time without creating a new object, thus providing flexibility and reducing redundancy.

5. Flattened Hierarchy for Overlapping Inheritance:

I flattened the inheritance hierarchy to resolve issues related to overlapping inheritance. This restructuring makes it possible to implement this type of inheritance and ensures the system is easier to utilize and maintain.

6. EnumSet for Agent Type Assignment:

I introduced an EnumSet to assign and manage agent types for overlapping. This approach provides a clear and efficient way to handle multiple agent types, ensuring that each agent's type is explicitly defined and easily manageable within the system.

Technologies used for implementation:

1. Spring Framework

Spring is a flexible framework for building Java applications. It simplifies the development process by providing comprehensive infrastructure support. It helps organize the code better, makes it easier to add new features, and keeps everything running smoothly.

2. H2 Database

H2 Database is a lightweight, fast, and open-source relational database management system written in Java. It's often used for development, testing, and embedded database scenarios. It's easy to set up and works quickly. H2 can run in memory or in a traditional server mode, making it versatile for various applications.

3. Thymeleaf

Thymeleaf is a tool for creating web pages in Java applications. It allows developers to create dynamic web pages using HTML, which can then be processed on the server to include data from the backend before being sent to the client. It works well with the Spring Framework and makes it straightforward to build web pages that look good and are easy to update. With Thymeleaf, you can create dynamic and interactive web pages without too much effort.

4. Lombok

Lombok is a Java library that helps you write less code by automatically generating common parts of your Java classes, like getters, setters, and constructors. It makes the code cleaner and saves time, making the development process faster and more efficient.

The design decisions and dynamic analysis, ensure that the Fatui Organizational System is capable of handling complex relationships and processes efficiently.